加微信：642945106 发送“ 赠送 ”领取赠送精品课程
发数字“ 2 ”获取众筹列表

22 | 系统案例：如何提高iTLB（指令地址映射）的命中率?

2020-01-15 庄振运

讲述：庄振运
时长 19:22 大小 15.53M

你好，我是庄振运。

我们今天继续探讨性能优化的实践，介绍一个系统方面的优化案例。这个案例涉及好几个方面，包括 CPU 的使用效率、地址映射、运维部署等。

开发项目时，当程序开发完成后，生成的二进制程序需要部署到服务器上并运行。运行这个程序时，我们会不断衡量各种性能指标。而生产实践中，我们经常发现一个问题：是指令地址映射的不命中率太高（High iTLB miss rate），导致程序运行不够快。我们今天就这个问题。

在我过去的生产实践中，针对这一问题，曾经采取的一个行之有效的解决方案，就是同时进行**二进制程序的编译优化**和**采用大页面的部署优化**。我下面就详细地分享这两个优化策略，并介绍如何在公司生产环境中，把这两个策略进行无缝整合。

为什么要关注指令地址映射的不命中率？

我们先来看看**为什么需要关注 iTLB 的命中率**。

在以往从事的性能工作实践中，我观察到 CPU 资源是最常见的性能瓶颈之一，因此提高 CPU 性能，一直是许多性能工作的重点。

导致 CPU 性能不高的原因有很多，其中有一种原因就是**较高的 iTLB 不命中率**。这里的 iTLB 就是 Instruction Translation Lookaside Buffer，也就是**指令转换后备缓冲区**。iTLB 命中率不高，就会导致 CPU 无法高效运行。

那么 TLB（转换后备缓冲区）又起到了什么作用呢？

我们知道，在虚拟内存管理中，内核需要维护一个地址映射表，将虚拟内存地址映射到实际的物理地址，对于每个内存里的页面操作，内核都需要加载相关的地址映射。在 x86 计算体系结构中，是用内存的页表（page table），来存储虚拟内存和物理内存之间的内存映射的。但是，内存页表的访问，相对于 CPU 的运算速度，那是远远不够快的。

所以，为了能进行快速的虚拟到物理地址转换，TLB（转换后备缓冲区）这种专门的硬件就被发明出来了，它可以作为内存页表的缓存。TLB 有两种：**数据 TLB**（Data）和**指令 TLB**（Instruction），也就是 iTLB 和 dTLB；因为处理器的大小限制，这两者的大小，也就是条目数，都不是很大，只能存储为数不多的地址条目。

为什么说 TLB 的命中率很重要呢？

这是因为，内存页表的访问延迟比 TLB 高得多；因此命中 TLB 的地址转换，比未命中 TLB 也就快得多。因为 CPU 无时无刻不在执行指令，所以 iTLB 的性能尤其关键。iTLB 未命中率，是衡量因 iTLB 未命中而导致的性能损失的度量标准。当 iTLB 未命中率很高时，CPU 将花费大量周期来处理未命中，这就导致指令执行速度变慢。

具体来讲，iTLB 命中和不命中之间的访问延迟，差异可能是 10 到 100 倍。命中的话，仅需要 1 个时钟周期，而不命中，就需要 10-100 个时钟周期，因此 iTLB 不命中的代价是极高的。

我们可以用具体的数据来感受一下。假设这两种情况分别需要 1 和 60 个时钟周期，未命中率为 1%，将导致平均访问延迟为 1.59 个周期，相比全部命中的情况（即 1 个周期）的访问延迟，足足高出 59%。

如何提高指令地址映射的命中率？

对于 iTLB 命中率不高的系统，如果能提高命中率，可以大大提高 CPU 性能并加快服务运行时间。我们在生产过程中实践过两种方案，下面分别介绍。

第一种方案，是优化软件的二进制文件来减少 iTLB 不命中率。

一般而言，根据编译源代码的不同阶段（即编译、链接、链接后等阶段），分别存在三种优化方法。这样的例子包括优化编译器选项，来对函数进行重新排序，以便将经常调用的所谓“热函数”放置在一起，或者使用 FDO（Feedback-Directed Optimization，就是基于反馈的优化）来减少代码区域的大小。

防止断更 请务必加
首发微信：17161436665

FDO 是什么呢？简单来说，就是把一个程序放在生产环境中运行，剖析真实的生产数据，并且用这些信息来对这个程序进行精准地优化。比如，可以确切地知道在生产环境中，每个函数的调用频率。

那么要如何进行二进制的优化呢？

我们可以通过编译优化来将频繁被访问的指令汇总到一起，放在二进制文件中的同一个地方，以提高空间局部性，这样就可以提高 iTLB 命中。这块放置频繁访问指令的区域，就叫**热区域**（Hot Text）。

在热区域的指令，它们的提取和预取会更快地完成。还记得我们在第 4 讲学过的**帕累托法则**吗？根据对许多服务的研究，帕累托法则在这里依然适用。

通常情况下，有超过 80%的代码是“冷的指令”，其余的是“热指令”。通过将热指令与冷指令分开，昂贵的微体系结构资源（比如 iTLB 和缓存）就可以更有效地处理二进制文件

的热区域，从而提高系统性能。

具体的二进制优化过程，包括以下三个大体步骤：

首先，是通过分析正在运行的二进制文件来识别热指令。我们可以用 Linux 的 perf 工具来达成此目的。你有两种方法来进行识别：可以使用堆栈跟踪，也可以使用LBR（Last Branch Record，最后分支记录）。LBR 比较适宜，是因为它的好处是能提高数据质量，并减少数据占用量。

其次，根据函数的访问频率，对配置文件函数进行排序。我们可以使用名为HFSort的工具，来为热函数创建优化表单。

最后，链接器脚本将根据访问顺序，优化二进制文件中的函数布局。

这些步骤执行完毕后的结果就是一个优化的二进制文件。我说明一下，如果这里面提到的工具你没有用过，也没有关系。这里知道大体原理就行了，当你真正用到的时候，可以再仔细去研究。

第二种方案就是采用大页面。什么是大页面呢？

现代计算机系统，除了传统的 4KB 页面大小之外，通常还支持更大的页面大小，比如 x86_64 上分别为 2MB 和 1GB。这两种页面都称为大页面。使用较大的页面好处是，减少了覆盖二进制文件的工作集所需的 TLB 条目数，从而用较少的页面表就可以覆盖所有用到的地址，也就相应地降低了采用页面表地址转换的成本。

在 Linux 上，有两种获取大页面的方法：

1. 手工：预先为应用程序预留大页面；
2. 自动：使用透明大页面，也就是THP（Transparent Huge Pages）。

THP，就像名字一样，是由操作系统来自动管理大页面，不需要用户去预留大页面。THP 的显著优点是**不需要对应用程序做任何更改**；但是也有缺点，就是**不能保证大页面的可用性**。预留大页面的方式，则需要在启动内核时应用配置。假如我们想保留 64 个大页面，每个 2MB，就用下面的配置。

```
1  hugepagesz = 2MB,  hugepages = 64
```

复制代码

我们在服务器上运行程序时，需要将相应的二进制文件加载到内存中。二进制文件由一组函数指令组成，它们共同位于二进制文件的文本段中，每个页面都尝试占用一个 iTLB 条目来进行虚拟到物理页面的转换。

如果内存页（比如 4KB）很小，那么对于一定大小的程序，需要加载的内存页就会较多，内核会加载更多的映射表条目，而这会降低性能。通常在执行过程中，我们使用 4KB 的普通页面。如果使用“大内存页”，页面变大了（比如 2MB 是 4KB 的 512 倍），自然所需要的页数就变少了，也就大大减少了由内核加载的映射表的数量。这样就提高了内核级别的性能，最终提升了应用程序的性能。这就是大页面为什么会被引入的原因。

由于服务器通常只有数量有限的 iTLB 条目，如果文本段太大，大于 iTLB 条目可以覆盖的范围，则会发生 iTLB 不命中。

例如，Intel HasWell架构中 4KB 页面有 128 个条目，每个条目覆盖 4KB，总共只能覆盖 512KB 大小的文本段。如果应用程序大于 512KB，就会有 iTLB 不命中，从而需要去访问内存的地址映射表，这就比较慢了。iTLB 未命中的处理是计入 CPU 使用时间的，所以等待访问内存地址映射的过程，就实际上浪费了 CPU 时间。

我们提出的第二个方案，就是使用大页面来装载程序的热文本区域。通过在大页面上放置热文本，可以进一步提升 iTLB 命中率。使用大页面 iTLB 条目时，单个 TLB 条目覆盖的代码是标准 4K 页面的 512 倍。

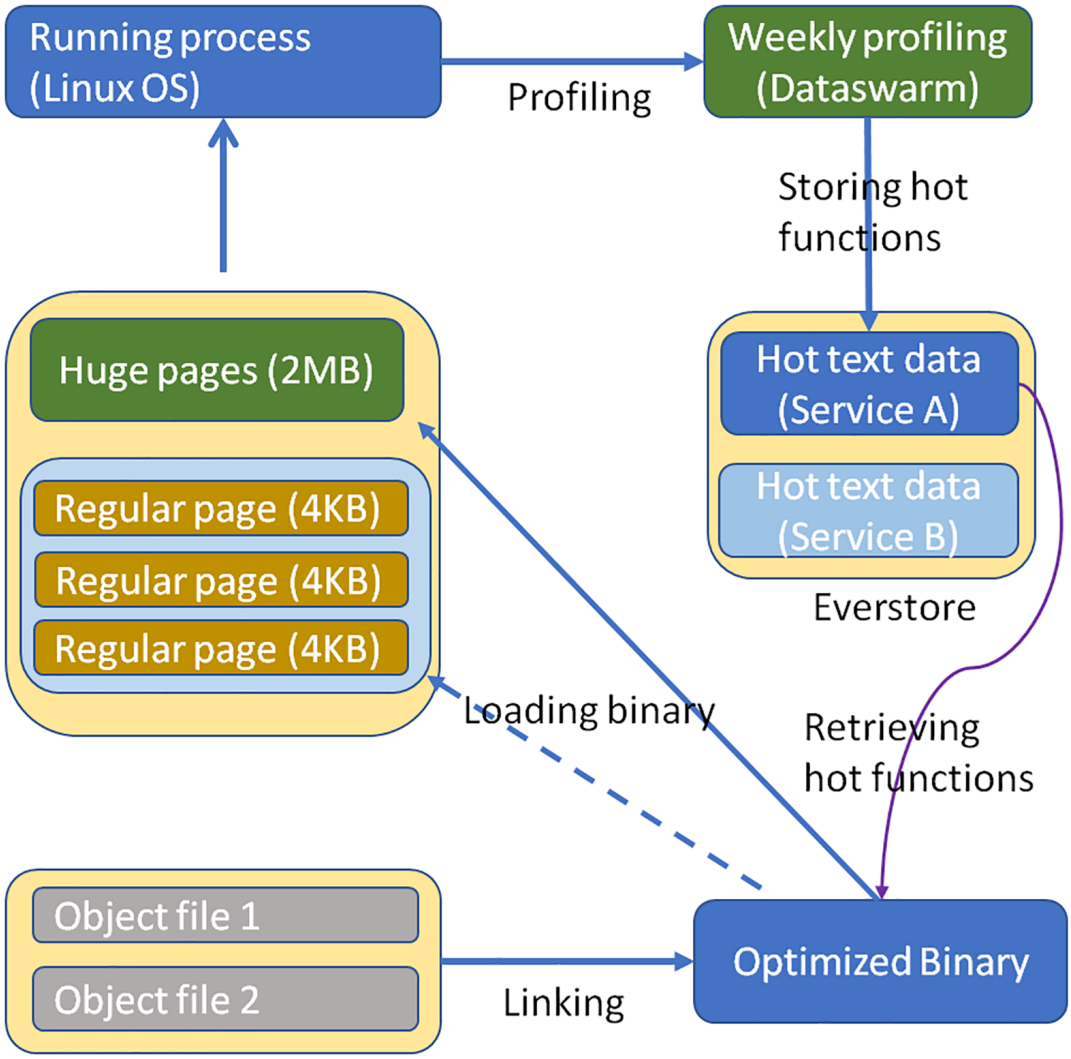
更重要的是，当代的 CPU 体系结构，通常为大页面提供一些单独的 TLB 条目，如果我们不使用大页面，这些条目将处于空闲状态。所以，通过使用大页面，也可以充分利用那些 TLB 条目。

如何获得最佳优化结果？

我们总共提出了两个方案，就是**采用热文本**和**采用大页面放置**。这两个其实是互补的优化方案，它们可以独立工作，也可以整合起来一起作用，这样可以获得最佳的优化结果。

采用热文本和大页面放置的传统方法需要多个步骤，比如在链接阶段，将源代码和配置文件数据混合在一起，并进行各种手动配置和刷新，这就导致整个过程非常复杂。这样的整合方案也就很难广泛应用到所有的系统中。

我们在生产中构建了一个流程，来自动化整个过程。这样，该解决方案就成为能被几乎所有服务简单采用的方案，而且几乎是免维护的解决方案。这个解决方案的流程图如下，整个系统包含三大模块：程序剖析（Profiling）、编译链接（Linking）和加载部署（loading）。



程序剖析

剖析模块显示在图的顶部。这个模块定期，比如每周执行一次数据收集作业，以剖析测量正在运行的服务。

Dataswarm 是我们曾经采用的数据收集框架，这是 Facebook 自己开发和使用的数据存储和处理的解决方案。这个作业剖析了服务的运行信息（例如，热函数），并且对配置文件进行控制以使其开销很小。最后，它会把分析好的数据发送到名为 Everstore的永久存储，其实这是一个基于磁盘的存储服务。

编译链接

在构建服务包时，链接程序脚本会从 Everstore 检索已配置的热函数，并根据配置文件，对二进制文件中的功能进行重新排序。这个模块的运行结果就是已经优化的二进制程序。

加载部署

加载服务二进制文件时，操作系统会尽最大努力，在大页面上放置热文本。如果没有可用的大页面，则放在常规内存页面上。

生产环境的性能提升

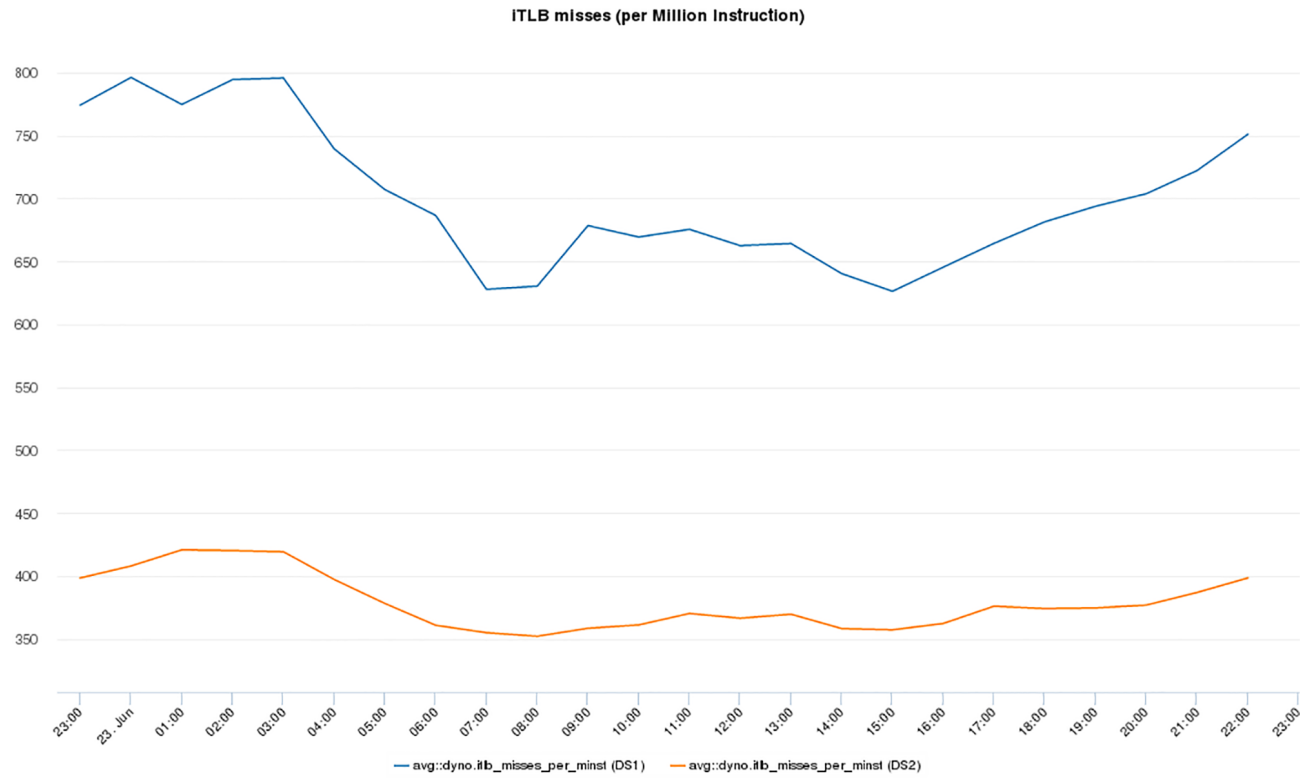
这样的解决方案实际效果如何呢?

我们曾经在 Facebook 的生产环境中广泛地采用这一优化策略。通过几十个互联网服务观察和测量，我们发现，应用程序和服务器系统的性能都得到了不错的提升，应用程序的吞吐量差不多提高了 15%，服务等待时间减少了 20%。

我们也观察了系统级别的指标。系统级别的指标，我们一般考虑主机 cpu 使用情况和 iTLB 不命中率。 iTLB 的不命中率几乎降低了一半，CPU 使用率降低了 5%到 10%。我们还估计，在这里面约有一半的 CPU 使用率降低是来自热文本，另一半来自大页面。

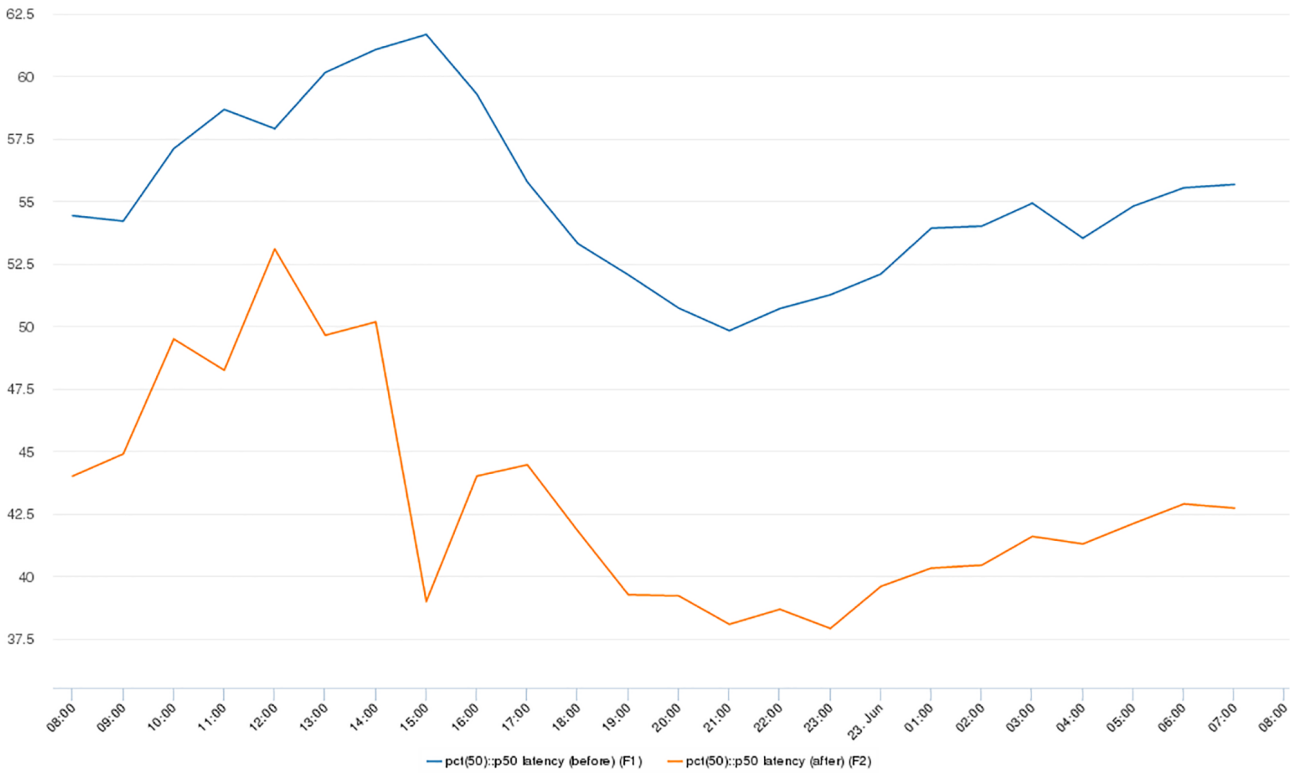
为了帮你更好的认识和体会性能的提升，我下面展示一个具体的互联网服务，在采用这个解决方案后的性能对比。

这张图显示了 iTLB 不命中率的变化。



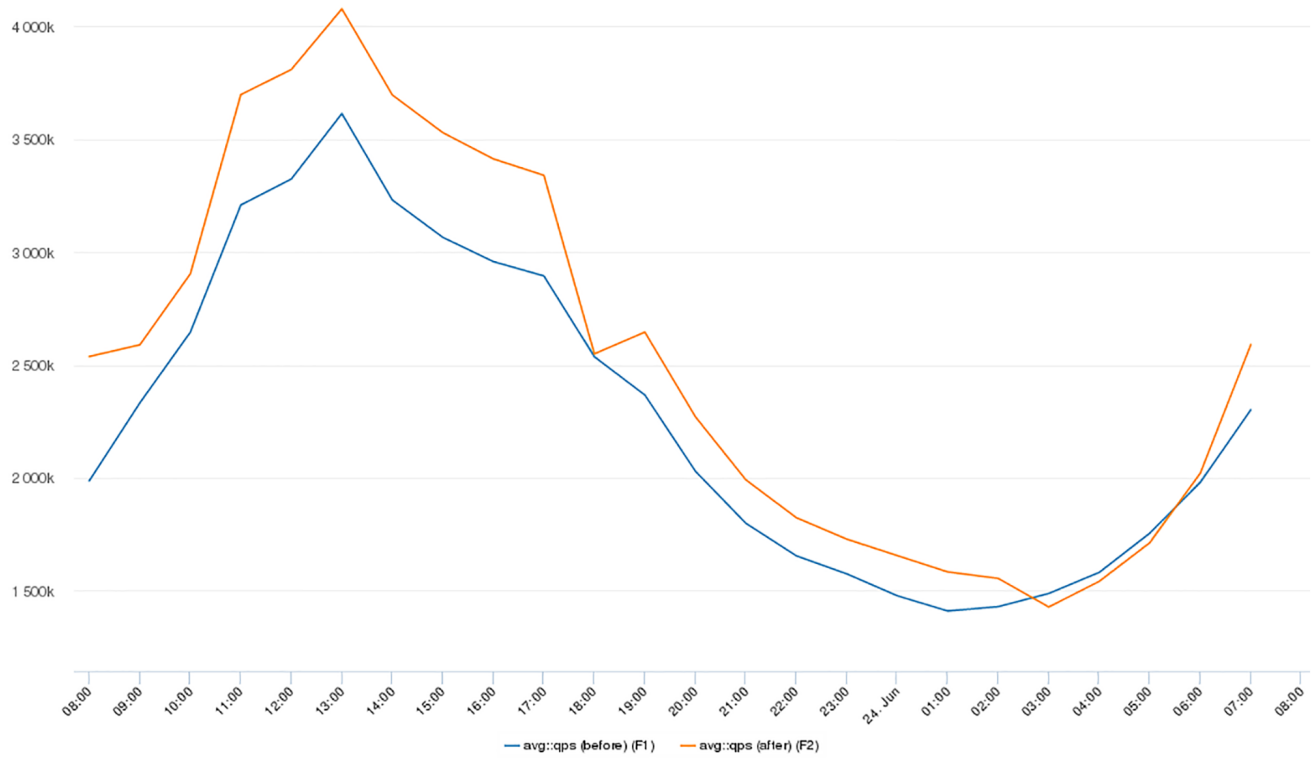
在应用该解决方案之前，iTLB 不命中率在峰值期间高达每百万条指令 800 个，如蓝色线表示。采用我们的解决方案优化部署后，iTLB 不命中率几乎下降了一半。具体而言，在峰值期间，最高的 iTLB 不命中率降低为每百万条指令 425 次，如黄线表示，相对优化以前下降了 49%，差不多是一半。

应用程序级别指标，显示在下图中。



蓝色曲线为优化前，黄色曲线为优化后。我们可以看到，应用程序请求查询延迟的中位数（P50）下降最多 25%，P90 百分位数下降最多 10%，P99 下降最多 60%。

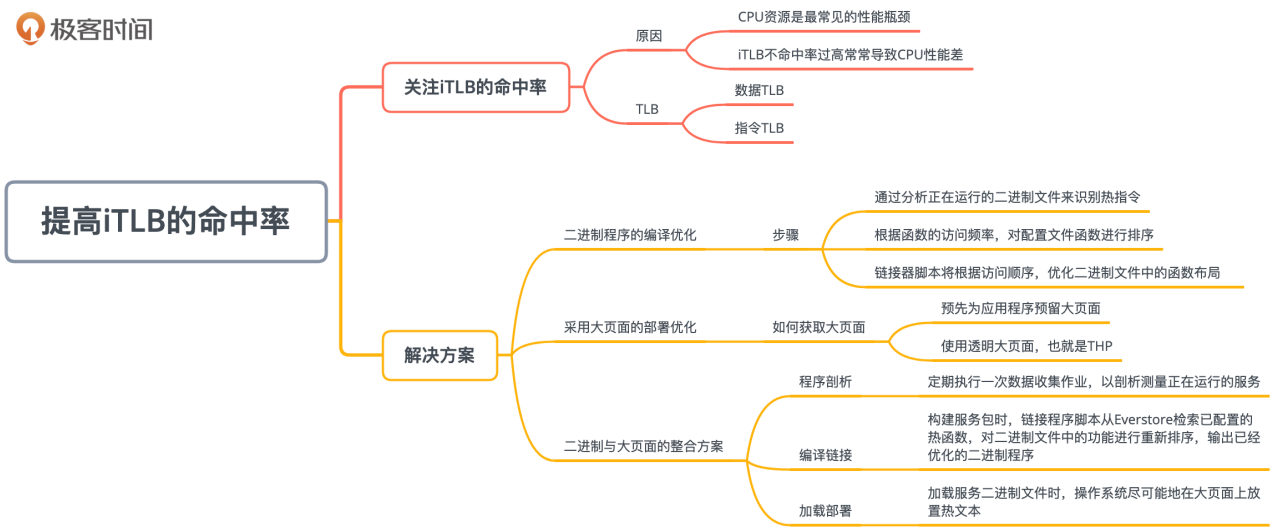
应用程序吞吐量（QPS）如下图所示，优化后增长了 15%，也就是说，峰值吞吐量从 3.6 万 QPS 增加到了 4.1 万 QPS。你可以看到，应用程序级别的吞吐量和访问延迟都得到了改善。



总结

这一讲我们讨论了如何有效地降低指令地址映射的不命中率太高的问题。

完整的解决方案包括两部分：**编译优化**和**部署优化**。对编译的优化，是进行指令级别的划分，把经常访问的指令放在一起，形成 Hot Text 区域。对程序部署的优化，是采用大页面。这两个部分在真正的生产环境中可以一起使用。



唐代的诗人张籍曾经鼓励一个出身寒门的朋友说：“越女新妆出镜心，自知明艳更沉吟。齐纨未足时人贵，一曲菱歌敌万金”。最后两句的意思是，大家追求的畅销东西，比如齐国的珍贵丝绸，恰如社会上横流的物欲，虽然贵重，但是见得多了也就不足为奇了。倒是平时不流行的东西，比如一首好听的采菱歌曲，更值得人称道看重。

操作系统的内存页面管理也有类似的道理，虽然普通的 4KB 页面容易管理，几乎每个程序都在用，大家已经习以为常；但是在某些部署场景下，大页面的使用会让系统性能大增，颇有点惊艳的效果。

思考题

Linux 操作系统中，一个常用的大页面，相当于多少个普通页面的大小？相对于普通页面，大页面有哪些优点，又有哪些缺点呢？

欢迎你在留言区分享自己的思考，与我和其他同学一起讨论，也欢迎你把文章分享给自己的朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　21 | CPU案例：如何提高LLC（最后一级缓存）的命中率?

精选留言 (4)

写留言

我来也
2020-01-15

长见识了，居然能优化到这种地步。

展开

1

kyleqian
2020-01-17

文中提到“链接器脚本将根据访问顺序，优化二进制文件中的函数布局”，请问这个脚本基于什么方式优化函数布局？链接过程可以自己控制吗？

展开

> 作者回复: Facebook把一些相关技术开源了，你可以参考下，包括HFsort和Bolt。 https://github.com/facebook/hhvm/tree/master/hphp/tools/hfsort； https://github.com/facebookincubator/BOLT

kyleqian
2020-01-17

请问在linux上不改系统配置，如何手工为应用程序指定大页面呢？

> 作者回复: 可以在程序里面调用，比如mmap。参考http://man7.org/linux/man-pages/man2/mmap.2.html

Linuxer
2020-01-15

请问文中的指标，什么样的取值算正常，什么样的取值算有问题呢？比如下面我抓的mysql的输出算正常吗？
117,811,688,709 dTLB-loads (100.00%)
804,095,370 dTLB-load-misses # 0.68% of all dTLB cache hits (100.00%)
772,145,652 iTLB-loads ...

展开

作者回复: iTLB的数据肯定不对（load 不可能小于miss)；应该是系统(CPU)的问题。
查了一下，好像有些CPU（包括Broad well）的iTLB-loads 被错误的映射到 ITLB_MISSES.STLB_HIT；　iTLB-load-misses映射到ITLB_MISSES.MISS_CAUSES_A_WALK。